4-21-2014

A Composite Interface for Bioinformatics Applications (CIBA)

Daniel F. Surdyk III

R·I·T

A COMPOSITE INTERFACE FOR BIOINFORMATICS APPLICATIONS (CIBA)

Daniel F. Surdyk, III

A Thesis Submitted in Partial Fulfillment of the Requirements
for a Degree of Master of Science in Bioinformatics

Department of Biological Sciences
College of Science

Approved By:
Dr. Michael V. Osier
Dr. Gary R. Skuse
Dr. Ronald P. Vullo

Rochester Institute of Technology
Rochester, New York
April 21, 2014

Page intentionally left blank

Abstract

Bioinformatics, and more importantly the general use of computers in the field of Biology, has become a mainstream exercise for today's biologist. So mainstream, that Bioinformatics classes are commonly offered to students in the standard Biology and Computer Science degree programs. This poses a problem for educators since some of the core Bioinformatics programs are still command-line based, requiring a deeper knowledge of computers than a standard biology student is expected to possess. The Composite Interface for Bioinformatics Applications, or CIBA for short, was created to address this issue and allow educators to focus on teaching how the algorithms work as opposed to teaching how the command line works.

Acknowledgements

First and foremost I would like to thank my advisory board. This would not have been possible without the direction and guidance they all provided. I would especially like to thank Dr. Michael Osier for reading my countless drafts, giving crucial input, and sticking with me throughout the years.

I would also like to thank the Rochester Institute of Technology and all the board members, educators and advisory boards that helped create the education you provide. You have given me an amazing, well-respected education, and I am proud to call myself an RIT Alum.

Finally, I would like to thank my family. For always pushing me to do my best. For never accepting "I can't". For the tireless nights you spent reading countless copies before they were good enough to send as a rough draft. And most importantly, for all the love and support you've given throughout the years that kept me going.

Table of Contents

Abstract iii

Introduction 1

Background 2

Materials and Methods 5

CIBA Architecture 6

Detailed Design 7

AbsCibaWizard: 7

AbsCibaWizardPage: 8

CibaJob: 9

plugin.xml: 9

User Interface 10

Results and Discussion 11

Existing Solution Analysis 11

GUI BLAST 12

GUI BLAST: Expandability 15

GUI BLAST: Comprehensiveness 15

GUI BLAST: Cost 15

Clustal X 15

Clustal X: Learnability 16

Clustal X: Simplicity 16

Clustal X: Informativeness 17

Clustal X: Consistency 17

Clustal X: Help Documentation 17

Clustal X: Error Handling 18

Clustal X: Expandability 18

Clustal X: Comprehensiveness 19

Clustal X: Cost 19

Geneious 19

Geneious: Learnability 20

Geneious: Simplicity 20

Geneious: Informativeness 20

Geneious: Consistency 20

Geneious: Help Documentation 21
Geneious: Error Handling 21
Geneious: Expandability 21
Geneious: Comprehensiveness 22
Geneious: Cost 22
BioEdit 22
BioEdit: Learnability 23
BioEdit: Simplicity 23
BioEdit: Informativeness 23
BioEdit: Consistency 24
BioEdit: Help Documentation 24
BioEdit: Error Handling 25
BioEdit: Expandability 25
BioEdit: Comprehensiveness 25
BioEdit: Cost 25
Bioclipse 26
Bioclipse: Learnability 27
Bioclipse: Simplicity 27
Bioclipse: Informativeness 28
Bioclipse: Consistency 28
Bioclipse: Help Documentation 28
Bioclipse: Error Handling 28
Bioclipse: Expandability 29
Bioclipse: Comprehensiveness 29
Bioclipse: Cost 30
Analysis Summary 30
CIBA- A Composite Interface for Bioinformatics Applications 30
CIBA: Clustal W Wizard 31
CIBA: PHYLIP, DNADist Wizard 35
CIBA: PHYLIP, Neighbor Wizard 36
CIBA: PHYLIP, Drawtree Wizard 37
CIBA: Current limitations, flaws, and issues 37
CIBA: Evaluation 38

CIBA: Learnability 38
CIBA: Simplicity 38
CIBA: Informativeness 39
CIBA: Consistency 39
CIBA: Help Documentation 39
CIBA: Error Handling 40
CIBA: Expandability 40
CIBA: Comprehensiveness 40
CIBA: Cost 41
Current State 41
Conclusion and Future Considerations 42
Works Cited 46

Introduction

The post-WWII era saw a surge in the academic availability of computers. By the 1960's, IBM had released the first high-level programming language FORTRAN, making it possible to program a computer without having detailed knowledge of its architecture (Hagen 2000). This was a crucial advance for the field of Biology, and in 1970 Saul Needleman and Christian Wunsch published the first computer-based algorithm on nucleotide and protein sequence comparison (Selzer, Marhofer and Rohwer 2008). Soon after came a spike in the number of databases, algorithms, and programs devoted to helping biologists organize, sort, and otherwise make sense of information they had gathered. Unfortunately, the limited computing power available at the time forced designs for such tools to be constrained by conflicting priorities: present the user with a clean, easy to navigate User Interface (UI), or dedicate every available CPU cycle to high speed and throughput. Many of the early Bioinformatics programs such as Clustal W, PHYLIP and BLAST focused on the latter, completely sacrificing an easy to use and understandable UI (at least as we understand them today). In fact, interaction with these tools occurred solely at the command line via "UNIX style commands ... with some suites having hundreds of possible commands, each taking different command options and input formats." As a result, a discipline-specific "single function black box" became the dominant design trend (Kumar 2005).

Through the years, our understanding of Bioinformatics has evolved. So too did the processing power of computers; eventually leading to an increase in size, complexity and popularity of these core Bioinformatics programs. While the reasons for this are outside the scope of this thesis, the fact remains that some of the most useful

Bioinformatics programs of today are still released as single function, command-line based black boxes (Sievers, Clustal: Multiple Sequence Alignment n.d.). Combined with the increased popularity of Computational Biology and multidisciplinary analysis, this poses a unique problem for educators: How can one efficiently teach the core principles and tools of Bioinformatics without having to first teach the intricacies of the command-line? It is the intent of this thesis to address the problem by creating a free, open source plugin for the Bioclipse application. The ultimate goal is to create a solution that will provide a seamless link between a backend CLI-based Bioinformatics program and a user-friendly front-end GUI so that students can focus on learning how to use the algorithms, not how to work the program.

Background

In silico scientific research, an activity once unheard of, is now an essential component in the arsenal of today's biologist. In fact, it is so common that businesses often rely heavily on initial *in silico* feasibility studies to assess whether a course of action could be fruitful prior to committing the significant labor and financial investment to the bench work required to develop the solution. Students either recognize the need for this computational skill, or are being encouraged to learn it. Additionally, it has been documented that entry-level students have a wide range of prior abilities and knowledge, ultimately influencing their ability to learn and problem solve in the classroom (Bransford, Brown and Cocking 2000). For example, one of the skills currently required to learn *in silico* analysis is the ability to understand and navigate the Command Line Interface (CLI). It also has one of the widest ranges of proficiency level, ranging from some students having mastered the concept, to others having never previously seen a

command line. For those lacking this experience, learning CLI-based programs such as Clustal W, BLAST, and the PHYLIP Suite becomes a much more difficult task.

Throughout the last century, much research has been done on the topics of pedagogy and human learning. In the late 1990s the *Committee on Developments in the Science of Learning* conducted a 2-year study on how people learn, shortly followed by an extension of that study by the *Committee on Learning Research and Educational Practice*. The results of both studies were summarized and published in 2000 in a book titled *How People Learn: Brain, Mind, Experience, and School* (Bransford, Brown and Cocking 2000). While this is not a discussion on the results and/or merits of that study, it may prove useful to summarize a few of their conclusions in order to gain a better understanding of why existing Bioinformatics solutions may be inadequate for an instructional setting.

According to the study, "one of the hallmarks of the new science of learning is its emphasis on learning with understanding" (Bransford, Brown and Cocking 2000). Essentially, they draw a distinction between what they call 'usable knowledge' and 'a mere list of disconnected facts'. Their argument is that learning information which has been organized into conditions and concepts makes it easily transferrable to situations outside the initial context in which the knowledge was gained. They conclude that this type of learning leads to a truer understanding of the topic as a whole.

Teaching inexperienced students the details of how to use the CLI so they can, at a minimum, execute a bioinformatics tool is akin to teaching them a 'list of disconnected facts.' In reality, it is nothing more than teaching the means to an end. To that point,

spending time explaining the CLI to proficient users is not, by absolute definition, 'usable knowledge' since they are already possessed the skill to begin with.

In addition to presenting an outright barrier to learning, the CLI is an inefficient means of instruction because it is inherently error-prone. In the article *Design Rules Based on Analyses of Human Error*, Donald Norman discusses the different types of human error and how the design of a computer program can either contribute to or minimize said error (Norman 1983). One of the major classes Norman cites is "mode error," which he defines as the result of a user executing a command appropriate for one mode of operation when the program is in an entirely different mode. An example of this in a GUI application would be trying to print a document using the "Ctrl-P" shortcut when the Save dialog is already being displayed. This class of error is fairly common, and is caused by "inadequate feedback and indication of the state of the system" (Norman 1983). With bioinformatics applications, use of modes is very common. An example of this error is seen in PHYLIP where non-applicable options are hidden from the menu, whereas in Clustal W all options are visible at all times. As such, any option visible in a PHYLIP application can have an impact on the resulting output, whereas this is not the case in Clustal W.

Another major contributor to errors with human-computer interaction is the "lack of consistency in the command structure" (Norman 1983). While most CLI commands are structured the same (the executable name followed by a list of arguments), it is left up to each application to determine how it will interpret that set of arguments. For example, Clustal W interprets arguments in the "-name=value" format while the PHYLIP suite requires that all input options be placed in one text file, with each new command on a

separate line. Then, the input file is specified at the command line using input redirection (i.e. "program.exe < inputFile.txt"), since none of the PHYLIP programs can accept parameters. Norman argues that the cause of this class of errors stems from unfamiliar users tending to derive interaction behavior for one program based on the prior experience of a related program. While all command structures have their own merit, their diversity and non-standardization demonstrates the root of why the CLI is not a good platform to use for teaching new technologies. In fact, hiding non-applicable options for the current mode prevents uneducated/inexperienced users from mistakenly thinking that a parameter will have an effect on the final outcome, yet showing such options can encourage experimentation, ultimately leading to a more thorough understanding of how the tool operates. Since each tool has the choice of which method it will employ, students are forced to learn these details up front before learning how to execute the most basic of analysis tasks. Alternately, instructors must walk unfamiliar students through every detail of the analysis process in lieu of teaching core concepts and letting them experiment for themselves. One of the ultimate goals of higher education is to promote self-sufficient learning, and it is the opinion of this author that the alternate option mentioned above is inconsistent with this goal. In addition, some of the most widely known bioinformatics tools in use today (Clustal W and PHYLIP) fail to promote this style of learning in a successful, consistent, and economically feasible manner appropriate for use in an educational institution.

Materials and Methods

The solution this thesis proposes is named *A Composite Interface for Bioinformatics Applications*, or CIBA for short. Like the name suggests, it is a

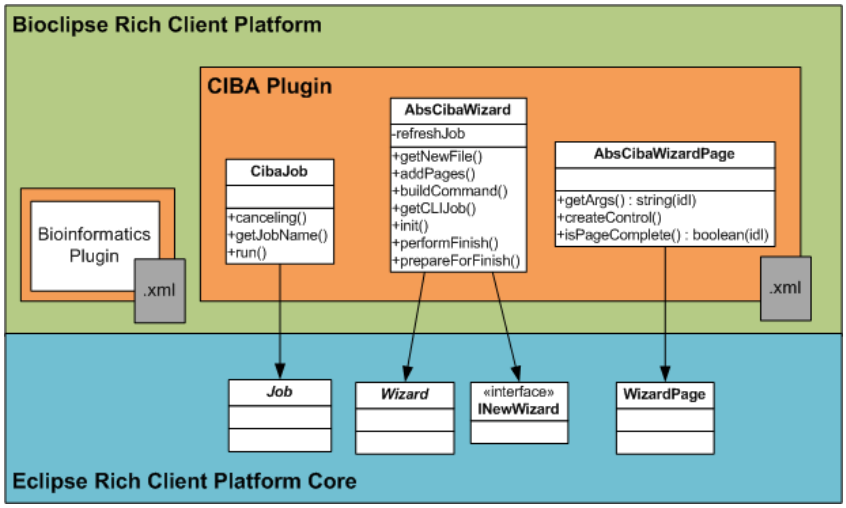
composite interface that defines an API for creating application wrappers that harness the power of the CLI, while hiding the details of the CLI interaction, to provide users with a consistent and intuitive GUI front end. CIBA uses the Bioclipse Rich Client Platform (RCP) as a foundation for implementation, is developed in Java, and is compatible with both Windows and Unix based operating systems. The Bioclipse RCP application was built on the Eclipse RCP framework, and the original papers pertaining to Bioclipse and Bioclipse 2 can be referenced if a more detailed description is desired (Spjuth, Alvarsson, et al. 2009).

CIBA Architecture

The CIBA plugin is a simple, single-tier application extension that uses the Model-View-Controller design pattern and draws upon the Wizard framework of the Eclipse API. The controller, and foundation for all CIBA wrappers, is the AbsCibaWizard. It extends the abstract Wizard class and implements the INewWizard interface. Each AbsCibaWizard implementation is then expected to contain one or more implementations of an AbsCibaWizardPage (the view and corresponding model) to handle displaying and gathering user input in an ordered fashion. The third major contributor that makes each CIBA plugin functional is the CibaJob, which serves as a unique extension of the core Eclipse Job by using the Apache Commons Exec[16] library for all of the command-line interaction. All user interaction occurs within the confines of the GUI wizard and its associated pages. The final step of the AbsCibaWizard will automatically schedule the CibaJob upon the user clicking "Finish" to complete the wizard. As for dependencies, CIBA uses BioJava for file and sequence parsing, making it dependent on the Bioinformatics plugin designed for Bioclipse 2, and uses the Apache Commons Exec library to handle all command-line interaction.

Detailed Design

The design for a CIBA wrapper begins with creating an implementation of the AbsCibaWizard class. The new wizard must be registered in the plugin.xml file that Bioclipse reads at startup, and it is expected to contain one or more CibaWizardPage implementations (see below). Eclipse has a number of useful pre-defined utility pages, such as the New Folder page, and their use inside CibaWizard wrappers is strongly encouraged.



AbsCibaWizard:

Concrete implementations of this class are designed to be the main control for each wizard. Expected tasks include adding/removing pages from the wizard, pre-populating page options, gathering all the options from each page, generating the correct command line argument(s), and creating the command line job. The following methods must be defined in concrete implementations:

- addPages() – Controls all pages visible in the wizard, and orders them appropriately.

- init(IWorkbench, IStructuredSelection) – Initializes the wizard with any selection-oriented information.
- buildCommand(Object …) – Build the command by gathering all options selected from each of the wizard's pages.
- prepareForFinish() – Called when the user clicks finish, this method should call buildCommand() and then prepare the wizard's CLI Job.
- getCLIJob() – Returns the CibaJob assigned to this particular wizard execution [typically instantiated in the prepareForFinish() method].

AbsCibaWizardPage:

Concrete implementations of this class are designed to be the view and model for each wizard's data. Expected tasks include initializing the page, maintaining its 'canFinish' status, and constructing the command line arguments the page is responsible for gathering. The following methods must be defined in a concrete class:

- createControl(Composite) – This method is used to initialize and draw all the components of the page. This should be auto-generated by a WYSIWYG editor such as Google's WindowBuilder Pro.
- getArgs() – Creates and returns a string representation of the arguments the particular page defines.
- isPageComplete() – Validate and indicate whether the information selected on the page is complete. The wizard framework uses this method to enable/disable the "Next" and "Finish" buttons.

CibaJob:

This class is not meant to be extended, though it can be if needed. It provides a number of different constructors to allow for varying uses which include running a simple command, redirecting input from a file, and changing the default timeout length. Currently, the default length for a run is 24 hours before the job will kill itself. Future updates will allow this to be a user-definable value, but for now individual implementations can change this via one of the constructors.

The run(IProgressMontior) method automatically creates an output console and links the command line output to the Bioclipse console being viewed. Instantiation should be done via one of the provided constructors, and as stated above scheduling occurs automatically in the CibaWizard as the last step to the user clicking “Finish”.

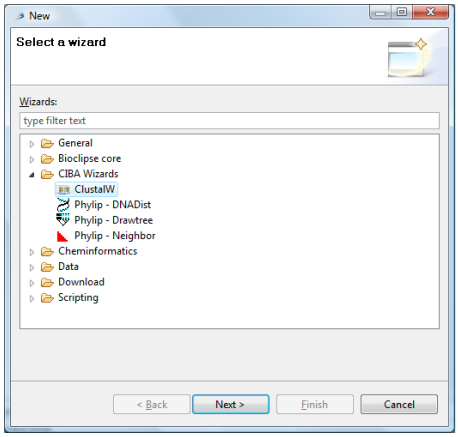
plugin.xml:

This XML file describes how the given plugin interacts with core Eclipse framework and is the key to Eclipse’s Lazy Loading abilities. It contains information such as new menu items that need to be displayed, which icons should be used for plugin-specific components, and what classes are mapped to particular functions implemented by the plugin. It should not be edited outside the Eclipse Plugin Development Environment, and any CIBA contributors not listed in this file will not be displayed in the menu system.

User Interface

The Bioclipse application itself is divided into two main areas: A navigator on the left and the working area on the right. The navigator lists current projects, folders, and files, and looks similar to what a Windows user would see when using the file browser. The working area flexes based on what the user currently has open and is editing.

The user interface for CIBA is only accessible from the workspace area on the left side of the screen and is intentionally quite basic. There are only two ways to create a new run (a single execution of one Ciba Wizard). The first is to navigate to the "File --> New" menu option and select the desired wizard from under the "CIBA Wizards" category. The second is to right-click anywhere in the workspace area ("Bioclipse Navigator") and select "New --> Other," followed by the desired wizard from under the "CIBA Wizards" category. Either choice will bring the user to the appropriate wizard where they can navigate through the pages, upload data, and select options as desired. As part of this thesis, Clustal W, and the DNADist, Drawtree, and Neighbor programs from the PHYLIP suite were implemented (details discussed in the following section).



The first step in all wizards created for this thesis is to select the output location for the run results, somewhere within the workspace area on the left. It is strongly recommended that future CIBA Wizards follow this same pattern, as it enables users to quickly define a location that is readily accessible and visible from the main application. Upon completion of the run, the wizards are designed to have the accessory application

save the results into the appropriate location and then refresh the view of the workspace area so users can see the new files as soon as they become available on the file system.

Upon completion of a run, the resulting files will be made available in the workspace area. The user is then free to analyze and/or use them as input to subsequence wizard executions.

Results and Discussion

Existing Solution Analysis

Design of early bioinformatics software solutions were heavily constrained by the limited computing power available at the time. As a result, many programs focused on the speed and accuracy of its results, to the detriment of the user interface. Over the last few decades, however, the raw processing power of computers and the ability to multi-thread and delegate specific tasks to other components (such as the GPU) has increased exponentially. Though this has led to some advances in algorithmic design, many of the most useful bioinformatics programs of today are still released in their CLI form (Sievers, Wilm, et al. 2011).

In addition, the last decade has brought about a new style of application in which individual CLI programs were converted to GUI applications, similar to what this thesis sets out to do. Some are simple wrappers of the single-function black box CLI applications they are intended to replace, and some are multi-function programs intended to provide combined access to otherwise disjoint programs. As is the nature of most software, this has been accomplished with varying degrees of success. Therefore, to determine characteristics of what a successful thesis solution should look like, five third-party applications were evaluated to determine how well they implemented their CLI counterpart. The analysis method chosen was to take a mixture of nine subjective and

objective evaluation categories and rate each application on a scale of 1 to 5. While a purely objective methodology is ideal, the diversity of functionality present between all five applications made this impossible. As such, the nine categories were separated into the Objective, Subjective, and Mixed subcategories. The completely objective categories included Help Documentation, Error Handling, and Comprehensiveness. The completely subjective categories included Learnability, Simplicity, and Cost. The mixed categories were Informativeness, Consistency, and Expandability. To help objectify the evaluation process as much as possible, the explanations in Table 1 were used throughout the analysis.

The investigation revealed a wide range of results from single function GUI-wrappers to fully developed GUI applications implementing multiple bioinformatics tools. The five applications selected for evaluation were GUI BLAST, Clustal X, Geneious, BioEdit, and Bioclipse. The purpose of each application is discussed below, followed by a detailed description of the evaluation results.

GUI BLAST

GUI BLAST is a program designed to provide an intuitive point and click front end to the most basic component of NCBI's BLAST suite. There are a number of advantages to using such a program. Creating a local BLAST query with GUI BLAST, as opposed to an online query on web-based UIs, eliminates the delay caused by multiple queries cued from other users around the world. Also, as the home page aptly points out, there is a decreased privacy risk with creating local BLAST query compared to sending it over an insecure internet connection as plain text. Lastly, it provides an easy

Table 1 - Rating Categories

	Category	Description
Subjective Categories:	Learnability (How "steep" the learning curve is)	1 – Unintuitive, difficult to learn 2 – Somewhat intuitive, still difficult to learn 3 – Average 4 – Intuitive, quick to learn 5 – Intuitive, quick to learn, design facilitates use
	Simplicity (after the initial learning curve)	1 – Obtuse to use / design impinged on use 2 – Functional, but design still could be improved 3 – Average 4 – Unified, easy to use 5 – Unified, easy to use, design facilitates use
	Cost	1 – Expensive, cost-prohibitive (>$500) 2 – Expensive, but not cost-prohibitive ($250 - $500) 3 – Average, acceptable cost ($100 - $250) 4 – Inexpensive, acceptable cost (<$100) 5 – Free Software
Objective Categories:	Help Documentation	1 – None 2 – Present, but some significant gaps 3 – Average, most items had help 4 – Full descriptions, just text 5 – Full descriptions, images, etc. Easy to access
	Error Handling	1 – None 2 – Present, but hidden 3 – Present, but not easily identifiable 4 – Graceful and automatic 5 – Graceful, automatic, and suggests a solution
	Comprehensiveness	1 – Bare bones functionality 2 – Limited functionality 3 – Full functionality of the program implemented 4 – Full functionality, plus additional features not present in CLI version 5 – Full functionality, 2 or more programs
Mixed Categories:	Informativeness	1 – Never informative 2 – Sometimes informative 3 – Mostly informative 4 – Always informative 5 – Intelligently informative (i.e. it is not too much information)
	Consistency	1 – Disparate implementations for similar functions 2 – Disparate implementation, but similar feel 3 – Somewhat unified presentation for similar functions 4 – Similar workflow for similar functions 5 – Identical workflow for similar functions
	Expandability (of functionality)	1 – Unable to be expanded 2 – Expandable, but with significant work 3 – Expandable with moderate effort 4 – Expandable with relative ease, but would leave cluttered implementation or effect existing implementation of functionality 5 – Easily / cleanly expanded without effecting existing functionality

and intuitive interface to the BLAST program, which is more user friendly than the command-line version. There are also a couple of disadvantages to using GUI BLAST with a local install of the BLAST suite. First, standard-build personal computers (where a local install of BLAST would run) are typically less powerful and slower than standard-build servers[1]. While this would mean slower BLAST execution for complex queries, one has to keep the intended audience in mind: users not proficient at the command line who, most likely, are not running overly complex queries in the first place. A second disadvantage is that GUI BLAST doesn't completely implement the full BLAST suite. Here again, however, the users most likely using all components of the BLAST suite would be experienced enough to use (and maybe even prefer) the CLI.

In addition to the above listed drawbacks, GUI BLAST has an overly complex install process that requires users to install three separate dependencies prior to installing the application itself. By the developer's own admission, he "hope[s] to make an easy to install .exe for Windows near the final release" (Digitales 2010). When GUI BLAST and appropriate dependencies were installed on both Windows and Unix based systems the application failed to start with the error "ImportError: No module named PythonCardPrototype". The GUI BLAST website did not indicate the need for a special configuration setting(s), so without the application starting the following categories could not be evaluated: Learnability, Simplicity, Informativeness, Consistency, Help Documentation, and Error Handling.

[1] It is possible to build a personal computer with an equivalent computing power to that of a server, but the typical user would likely have an off the shelf system.

GUI BLAST: Expandability

For being an open source program, it was given a 2/5. The graphical interface was solely limited to BLAST functionality, and it lacks documentation indicating how one would build upon what already exits. In addition, at the time of this writing, GUI BLAST was still in the pre-alpha stage of development, meaning it isn't complete enough for general use and/or beta testing.

GUI BLAST: Comprehensiveness

For implementing an incomplete version of one of the most basic BLAST tasks, it was given a 1/5. Furthermore, the download site indicated that the functionality already implemented still needed further polishing (Digitales 2010).

GUI BLAST: Cost

Being a free, open source project, GUI BLAST was given a full 5/5.

Clustal X

Clustal X is a "windows interface for the widely-used progressive multiple sequence alignment program Clustal W" that is "easy to use, [provides] an integrated system for performing multiple sequence and profile alignments, and analyzing the results" (Thompson, et al. 1997). Clustal X has all the standard functionality contained in its CLI counterpart Clustal W, plus additional features such as color-coded visualization of the final alignment, the ability to copy/paste sequences, modify the input order, and select a subset of sequences from which to start another alignment (Thompson, et al. 1997). One major disadvantage to Clustal X is that it's another single function black box, making expansion a difficult task. Originally developed in the C programming language, Clustal W and Clustal X version 2.0 were re-developed in 2007 using C++ (Larkin, et al. 2007). Since the time this analysis was completed, the Clustal authors have released

Clustal Omega: another re-developed version of the Clustal program touting better scalability and use of the multi-processing power of today's standard personal computers. Clustal Omega, first released in June 2011, is still released as command-line only (Sievers, Clustal: Multiple Sequence Alignment n.d.). In addition to Clustal X being a single function program, the fact that the core Clustal functionality has been redeveloped three times since its original inception without improving or rewriting Clustal X indicates that it is probably not robust enough to support the type of expansion required to solve the need this thesis seeks to address.

Clustal X: Learnability

In the category of learnability, Clustal X scored a rating of 4/5. Most actions were intuitive to pick up, and easy to use. However, setting the alignment parameters was slightly confusing and took some time getting used to since each set of options had to be accessed through a different menu/submenu. In addition, one had to set the alignment parameters prior to selecting the "Do Full Alignment" option, since it didn't prompt at the time of execution as one would have expected.

Clustal X: Simplicity

In the category of Simplicity, the rating was 3/5 for a variety of reasons. The program was relatively simple to use, and followed the same general flow used in the command line (i.e. upload the sequences, set the parameters, and then perform the alignment). However, Clustal X lost a point for not having implemented file filters for uploading common sequence file formats. This could potentially make it difficult to upload sequences if a user didn't already know the supported formats, thus violating the "facilitating use" requirement to get a 5/5. In addition to having a confusing design for parameter selection, it lost another point for not implementing a 'stop mid-alignment'

option. This was because the only way to stop/restart a long-running alignment (useful if one were to forget to set a parameter before starting) is to forcibly close Clustal X, restart the application, and reenter all of the sequences and parameters, thus violating the "unified" presentation requirement to be a 4/5.

Clustal X: Informativeness

In the category of Informativeness, Clustal X was given a 2/5 for being sometimes informative. For example, after successfully completing an alignment, the application only indicated where one output file was saved despite giving the ability to define different locations for each type of output file. For an application that would likely execute for hours, it could become problematic having to remember the location of individual output files. In addition, when error messages were displayed, they were often quite vague (see Error Handling section for examples).

Clustal X: Consistency

In the category of consistency, the program scored a 3/5. There were some popup windows that implemented "Ok" buttons in a non-standard location and in a different case than the rest of the application. For example, all parameter setting popups had the "OK" button (all upper case) in the upper left, while the help section popups had the "Ok" buttons (note the different case) in the lower left, yet warnings were implemented with the "Ok" buttons in the industry-standard lower right corner of the popup. While this seems like a trivial detail, it can be a source of frustration for new users.

Clustal X: Help Documentation

The Help Documentation for Clustal X scored a 2/5. While it implements a GUI help system, it is subject to the same design flaw that the parameter-setting implementation suffered from: separate popup menus for each item. Adding to the flaw,

each popup was modal, meaning that the help menu must be closed before returning to the program and completing the task at hand. Although the preferences implementation is also modal, there is a distinct advantage to displaying help concurrent with the program so a user can read and click at the same time. In fact, it's quite possible a user would open Clustal X twice, side by side – once to interact with the parameters, sequences, and options, and once for the sole purpose of opening a help window beside the first open instance. Contributing to such a low score was also the fact that the help system didn't add additional value to the help files already part of the main Clustal W program. Typically one would expect the help system for a GUI application to contain screen shots with key areas circled or otherwise identified for clarity. However, Clustal X has no images/screen shots in the documentation.

Clustal X: Error Handling

Error Handling was another weak point for Clustal X, and scored a 2/5. While present, it was difficult to interpret the meaning behind the error message. For example, two non-sequence files were uploaded (a foreseeable mistake given that no file filtering was implemented), and the first error stated "There were empty sequences in the input file". Since the uploaded file was actually a PDF document, the message was true but misleading. On the second erroneous sequence file, the message was a bit clearer: "No sequences in the file: bad format?" It is good practice to suggest a solution for any error condition (Microsoft n.d.), which the first message failed to do and the second message only hinted at.

Clustal X: Expandability

In the category of Expandability, Clustal X was given the lowest score possible: a 1/5. It is another "single function black box", and the whole design is wrapped around

the Clustal W tool, making it nearly impossible to expand Clustal X to implement the functionality of other programs without a significant redesign. Furthermore, there were enough design flaws that if any changes were to be made, more value would be gained by fixing what is currently suboptimal than by implementing new functionality.

Clustal X: Comprehensiveness

In the category of comprehensiveness, Clustal X was given a 4/5. It added functionality to the default Clustal W application, however as indicated above Clustal X only implemented the one program.

Clustal X: Cost

In the category of Cost, Clustal X scored a full 5/5 for being an Open Source, free application.

Geneious

Geneious is a cross-platform, ultra-powerful and easy to use bioinformatics software suite useful in manipulating and exploring biological data (Biomatters Ltd. n.d.). It has a very large feature set ranging from simple sequence alignment and primer design to all-out Next Generation Sequence Assembly. It even aids in the publication process by helping gather the high resolution images/charts/graphs/etc necessary for inclusion in a paper (Biomatters Ltd. n.d.). However, there is one main drawback: The product is a commercially based source of revenue for the company, and not all features are available in the free version. The academic cost associated with the "Pro" functionality (which includes MUSCLE, Clustal W and PAUP* integration, to name a few) starts at $395 for a student license and goes up to $1995 for a concurrent academic license.

Geneious was written in Java and has a freely available API for developers that desire having a customized solution. While this API encourages expansion, it is only enabled for the Pro version, the purchase of which already includes many common bioinformatics programs.

Geneious: Learnability

In the category of learnability, Geneious scored the highest rating of all existing programs: a 5/5. When using the application, it was very well designed, intuitive, and easy to use. The design definitely facilitated one task flowing easily into the next.

Geneious: Simplicity

In the category of Simplicity, the rating was 4/5, as the program was relatively simple to use once the learning curve had been overcome. Again, everything in the application seemed well designed and intuitive to access. From the eclipse-style workspace area (or "Sources View") on the left side, to the working area on the right ("Document Table" and "Document Viewer"), everything was well organized.

Geneious: Informativeness

In the category of Informativeness, Geneious was given a 4/5. It was always informative, but sometimes too much so: Upon double-clicking anything in the Document Table, it would re-open in a popup window. While this feature may be useful for a side-by-side comparison, it quickly became cumbersome and overwhelming in a single-monitor system.

Geneious: Consistency

In the category of consistency, this program scored 4/5. Most of the main actions followed a standard 'wizard style' guidance, and all popup windows had the same Navigator/Table/Viewer layout. Although the Document Viewer in Geneious has the

ability to display a variety of different viewers, all have the same basic layout: sequence information on the left and options to change or otherwise work with the sequence data on the right.

Geneious: Help Documentation

The Help Documentation category for Geneious scored a 5/5 for having a fully implemented, advanced tutorial system that includes screen shots. This system includes a version specific 162-page online manual that is one click away in the Help menu, as well as a context-sensitive help system that changes the help content based on what the user is currently interacting with. The most impressive component of the Help is an integrated form to submit a bug/feedback report with the ability to attach additional information (screen shots, the imported file, and/or an external attachment) with the simple selection of a checkbox.

Geneious: Error Handling

Error Handling was clean and nicely implemented as well, deserving the full 5/5. Though it was difficult to cause an error condition, attempting to import a sequence from a non-sequence file induced a popup that clearly indicated no sequence could be found. The message was clear, concise, and suggested that the user submit a bug report if deemed appropriate.

Geneious: Expandability

In the category of Expandability, this program was given a 5/5. It has public API based on Java and does allow for personalization and expansion with what appears to be only a moderate amount of work. It is worth noting that the public API could not be tested since it is only enabled with a paid license.

Geneious: Comprehensiveness

In the category of comprehensiveness, Geneious scored the highest rating possible (5/5) for having implemented multiple bioinformatics applications. It is worth noting that a 4/5 was considered since most of the applications are only available in the Pro version, however cost is another category so in the spirit of complete objectivity Geneious met the requirements for the full 5/5.

Geneious: Cost

In the last category, Cost, Geneious scored a 2/5, representative of it being an expensive but not cost-prohibitive program. While they offer a free version as well as discounted student pricing plans ranging from monthly to lifetime, all of the options require some form of significant commitment or decreased functionality.

BioEdit

BioEdit is "a mouse-driven, easy-to-use sequence alignment editor and sequence analysis program." It was developed as "a single program that can handle most simple sequence and alignment editing and manipulation functions" (Hall 2005). It has a wide array of functionality ranging from support for common sequence file formats (Genbank, Fasta, PHYLIP 3.2 & 4, etc), to the ability to run the full suite of BLAST programs. It can even be configured to interface with custom, user-defined Accessory Applications. Unfortunately, there are a few significant drawbacks to BioEdit. Configuring how it should use an Accessory Application is not an easy task. There is a complex and overloaded window (Figure 1) one must navigate to initially configure the application. Once that is complete, there is the potential that a user will still be required to enter a CLI command or two at the time of execution, thus defeating the purpose of creating a GUI

wrapper. Second, and more important, BioEdit is a dead project whose support ended at Windows XP. According to the website, which was last updated on June 27, 2007, "BioEdit is no longer being maintained, and the documentation is out of date" (Hall 2005). So, even if one did successfully implement an accessory application, there is no hope of it being included in a future release of BioEdit.

BioEdit: Learnability

In the category of learnability, BioEdit scored a rating of 3/5. When using the application there were some concepts and actions that weren't intuitive or easy to learn. In addition, the menus seemed to be overloaded with too many options (for example the "Sequence" menu had 32 separate items, 15 of which were sub-menus that expanded further on hover-over). This made it difficult to remember where a particular item/function was for future reference.

BioEdit: Simplicity

In the category of Simplicity, the rating was 4/5. The program was relatively simple to use once the learning curve had been overcome, however crowded/overloaded menus did not facilitate use, the requirement that set a score of 4 apart from a full 5. In addition, the "Add Accessory Application" popup was overly complex, as illustrated in Figure 1.

BioEdit: Informativeness

In the category of Informativeness, BioEdit was given a 4/5 for being very informative. It was not given a full 5/5 because the information displayed was obtrusively relayed via multiple unnecessary popup menus. Similarly, executing a Clustal alignment via each of the external application functions was informative, however

one gave real-time feedback, while the other seemingly froze BioEdit and gave all output at once after the alignment had been completed.

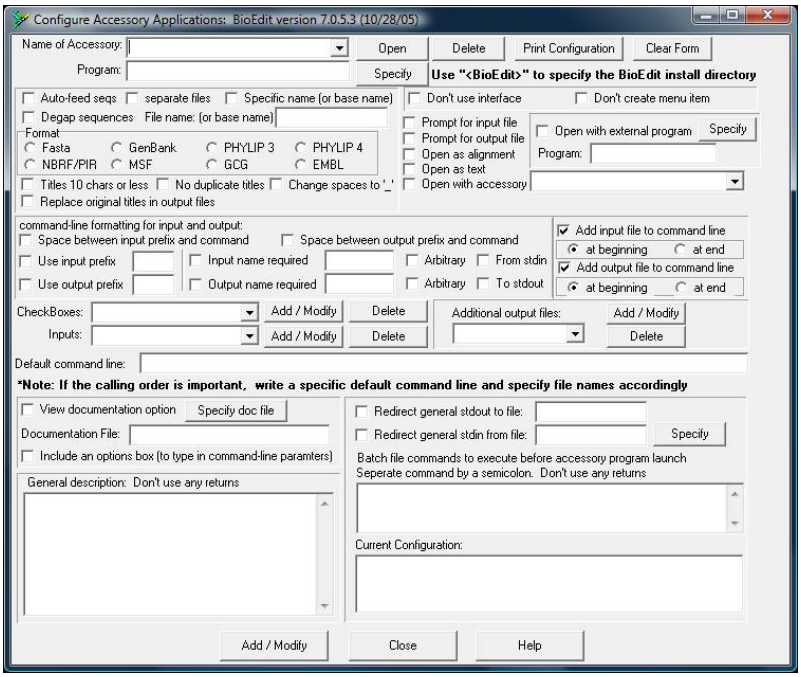


Figure 1 - BioEdit Accessory Application Configuration

BioEdit: Consistency

In the category of consistency, this program scored 2/5. As mentioned above, there were two menu items that could be used to create a Clustal alignment. While this is normally beneficial, the set of parameters one could configure differed between each of the menus, and the style of user feedback during the execution ranged from real-time to a temporarily frozen application.

BioEdit: Help Documentation

In the Help Documentation category, BioEdit scored a neutral 3/5. The program indicates that Help is present, and it was verified to work in Windows XP. However the help functionality does not work in Windows Vista, likely because Vista was first

released to the public in early 2007 while support for BioEdit ended around June 2007 (Hall 2005).

BioEdit: Error Handling

Error Handling was another neutral 3/5 for being present yet obtrusive. Again, BioEdit overuses popup notifications instead of a cleaner status bar or error bar implementation.

BioEdit: Expandability

Regarding Expandability, this program was given lowest score possible: a 1/5. Active support ceased by June 2007 (Hall 2005), making it a dead project. Additionally, the menus are cluttered and popup notifications are overused, so adding additional functionality would only exacerbate existing drawbacks. As such, the only solution to adding more functionality would have to include an entire user interface redesign, the creation of a new toolbar and status bar, and the simplification of the menus and preferences window.

BioEdit: Comprehensiveness

In the category of comprehensiveness, BioEdit scored a 4/5, the highest rating of all five applications analyzed. While it can interact with multiple accessory applications, not all are fully implemented. For example, BioEdit implemented only 6 of the 31 programs the PHYLIP package is composed of, and only implements a small subset of the options available with Clustal W.

BioEdit: Cost

In the category of Cost, BioEdit scored a 5/5 for being an Open Source, free application.

Bioclipse

Bioclipse is an Eclipse-based Rich Client Platform (RCP) designed for use in advancing life sciences work. It has a wide array of functionality including a Cheminformatics suite, a limited Bioinformatics feature set, and "many features that simplify pharmacological research and drug discovery" (Bioclipse Features n.d.). Although the Bioinformatics tools are heavily dependent on the open source BioJava project, and are currently limited to downloading biological sequences from Web Services and graphical editing of sequences in the sequence viewer, there is a lot of potential for expansion and development. In fact, once the SWT learning curve had been overcome, developing a plugin for any Eclipse based application is exceptionally simple with the *Eclipse IDE for RCP and RAP Developers*. The language of choice for Bioclipse is Java, and it is still actively developed and maintained. Two main strengths of Bioclipse include its dependence on BioJava, which makes up-to-date support for bioinformatics as simple as updating a JAR file, and its use of Eclipse as the main foundation, which brings with it the inherent stability of a mature, well-vetted project. The dependence on Eclipse, however, can be problematic. Because Eclipse is on its own development cycle, any bugs discovered in the core functionality have the potential to adversely affect Bioclipse and are invariably at the mercy of the Eclipse development team to be fixed.

It is important to note that some critics may view heavy dependencies (such as BioJava) as negative. However, when integrated correctly, dependencies can actually be of great benefit to an application. For example, Eclipse (and inherently Bioclipse) have this integration already well-defined since they use the Open Services Gateway initiative

(OSGi) framework as the foundation of their core plugin model. For RCP developers, this means simply wrapping each JAR dependency in its own bundle, adding that bundle to the dependency list in the desired plugin project, and then using the appropriate API during integration. Since the framework allows smooth integration of pre-built and pre-tested sub-systems, this methodology provides a number of benefits, including decreased production time and reduced development cost. In addition to eliminating the need to 'reinvent the wheel', use of the OSGi's explicit interface declaration principles decreases namespace conflicts and allows two bundles of an application to use different versions of the same dependency in the same JVM runtime (Paulin 2010).

Bioclipse: Learnability

In the category of learnability, Bioclipse scored a rating of 4/5. Most actions are moderately intuitive, and Bioclipse has a fairly detailed Welcome page that explains much of the basic functionality. The Welcome page also has the ability to display a useful step-by-step tutorial for the cheminformatics plugin, and includes some sample data useful for exploring all the cheminformatics plugin has to offer.

Bioclipse: Simplicity

In the category of Simplicity, the rating was 3/5. It was relatively simple to use once the learning curve had been overcome. However some items were confusing in the functionality of the Jmol menu because they had no effect when activated. Further investigation showed that the menu items functioned properly when one or more atoms were selected in a chemical structure. Combined with the fact that other menu items were appropriately enabled/disabled based on the state of the application, it was concluded that the defect is specific to the cheminformatics plugin and not the core Eclipse RCP application.

Bioclipse: Informativeness

In the category of Informativeness, Bioclipse was given a 2/5 for being only somewhat informative. There is a feedback area built into the lower right corner of the application that seemed to be used only occasionally. However being in the lower right makes it difficult to see unless one already knew it is there. In addition, when the Jmol menu items were not functioning properly, no error messages were displayed.

Bioclipse: Consistency

In the category of Consistency, this program scored 3/5. As part of a RCP application, a number of standard menus are already implemented that just need to be extended. Bioclipse did this in the navigator view by implementing a couple context-sensitive "New" items, adding a custom "Jmol" menu at the top of the root menu bar, and adding a content specific right-click popup menu in the cheminformatics perspective. However, the contents of the Jmol menu did not match the same structure as the right-click popup menu, even though they contain roughly the same functionality. In another area, the Bioclipse implementation of wizards and preference selections follow the standard Eclipse framework and all have the same look, feel and flow.

Bioclipse: Help Documentation

The Help Documentation for Bioclipse scored a full 5/5. The program implements a Welcome page, a fairly detailed help section, and a step-by-step tutorial for walking users through creating new chemical structures. As expected, the help functionality worked in both Windows and Unix-based systems.

Bioclipse: Error Handling

In the category of Error Handling Bioclipse was given a neutral 3/5. The functionality was present but difficult to understand for a non-technical user. For

example, when trying to generate the Balloon 3D image from a 2D ".cml" chemical structure file, the error was "'Balloon 3D conformer generation' has encountered a problem." Upon expanding the details, the reason listed was:

> java.io.IOException: Cannot run program "C:\Users\Daniel\Desktop\bioclipse_082110\bioclipse.balloon\plugins\net.bioclipse.balloon.linux\exec\balloon": CreateProcess error=193, %1 is not a valid Win32 application.

While a technical user would easily be able to tell that Bioclipse was using a Unix-based program on a Windows system, a non-technical biologist/chemist would likely miss this. In either case, Bioclipse provided no way to adequately set the install location of the Balloon 3D executable, making this an unrecoverable error.

Bioclipse: Expandability

Regarding Expandability, this program was given highest score possible: a 5/5. It is an open-source, easily expandable program that has a very well-developed API. In fact, the core of what Bioclipse was built on (Eclipse RCP) was specifically designed for modularity and expandability.

Bioclipse: Comprehensiveness

In the category of Comprehensiveness, Bioclipse was given a 5/5. It has multiple fully implemented applications spanning a couple of disciplines. Although not all applications are applicable to Bioinformatics, and most of the Bioinformatics functionality listed on the Bioclipse Developer Wiki was just a proof of concept (Spjuth, e-mail message to author 2010), the fundamental characteristics defining a 5/5 in Comprehensiveness were met.

Bioclipse: Cost

In the category of Cost, Bioclipse scored a full 5/5 since it is an Open Source and free application.

Analysis Summary

Of the five applications analyzed, Geneious performed the best, scoring a total of 35/45. Bioclipse was the next highest performer at a 34/45, and due to its high Expandability and cost ratings was chosen to be the core application this thesis was built upon. GUI BLAST performed the worst at 8/45 due to the difficulties installing and running the application causing 6 of the 9 categories to go unrated. Individual application results for each category analyzed can be seen in the table below.

Table 2 - A summary of the results

	GUI BLAST	Clustal X	Geneious	BioEdit	Bioclipse
Learnability (5)	N/A	4	5	3	4
Simplicity (5)	N/A	3	4	4	3
Informative (5)	N/A	2	4	4	2
Consistency (5)	N/A	3	4	2	3
Help Documentation (5)	N/A	2	5	3	5
Error Handling (5)	N/A	2	5	3	3
Expandability (5)	2	1	5	1	5
Comprehensiveness (5)	1	4	5	4	4
Cost (5)	5	5	2	5	5
Total (45)	8	26	39	29	34

CIBA- A Composite Interface for Bioinformatics Applications

In addition to the core framework for CIBA, four individual wizard implementations were also completed from this thesis. They include the Clustal W 2.12, PHYLIP-Drawtree, PHYLIP-DNADist, and PHYLIP-Neighbor wizards. With the core functionality in place, it will be very straightforward to add new CIBA Wizards with

relative ease, using the existing four implementations for guidance as needed. In fact, it was this author's experience that the most time consuming part to creating CIBA functionality was mapping new GUI components to their respective command line arguments.

CIBA: Clustal W Wizard

The design for wrapping Clustal W 2.12 into a CIBA Wizard was the most complex of all three implementations. It involveed one main Wizard with three individual WizardPages. The AbsCibaWizard implementation is called the ClustalWizard, and the three AbsCibaWizardPages are the SeqInputPage, the PairwiseAlignPage, and the MultiAlignPage. Upon entering the wizard, the first screen presented to the user is a Folder Selection page. A parent folder inside the user's workspace must be defined, and should typically be a root project, although that is not enforced at this time. The next page, the SeqInputPage, is where individual sequences can be uploaded, or two previous profiles can be merged (depending on the user's workflow). The wizard implementation disallows completion of the run if appropriate input is not entered.

New features implemented in the ClustalWizard not available with the command line version include the ability to preview individual sequences (disabled for sequences larger than 500,000 bases due to performance issues), the ability to remove individual sequences from the list, and improved error detection when corrupt files are selected for import.

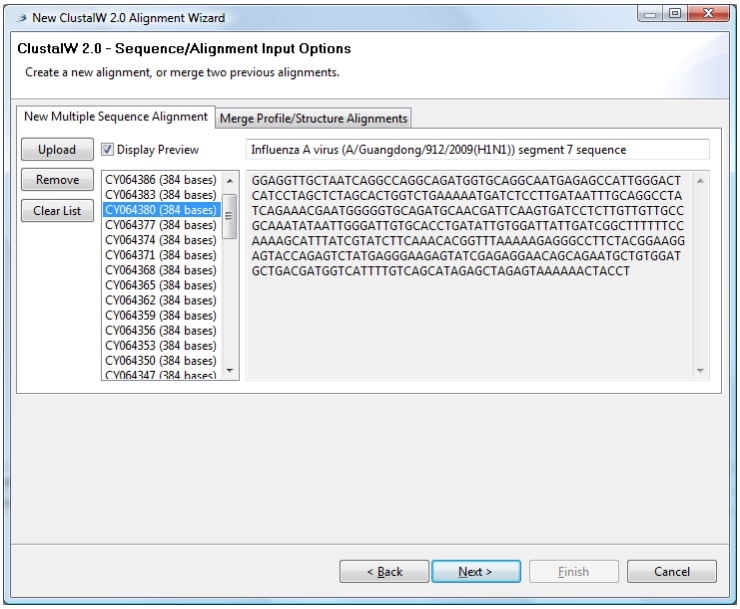


Figure 2

CIBA: Normal Sequence Input

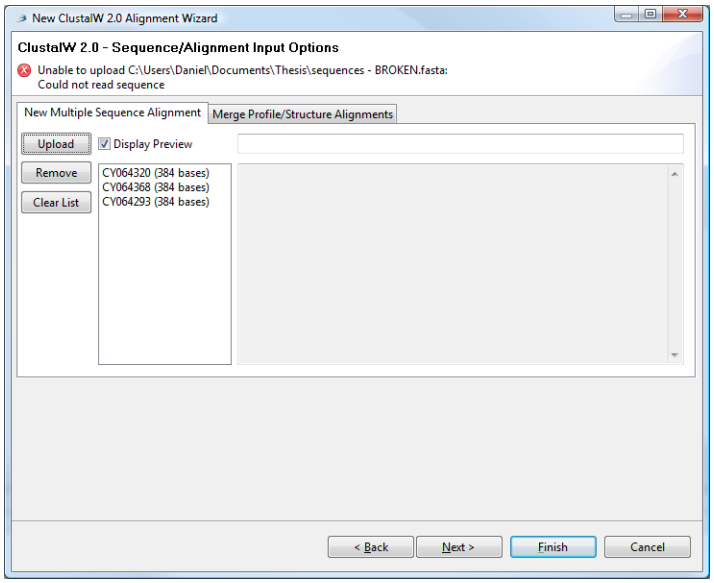


Figure 3

CIBA: Sequence Input from a Corrupt File

The next wizard page allows the user to select the Pairwise Alignment parameters and the final output options, such as enabling PHYLIP output. If the default options are sufficient, then this screen can be bypassed by clicking the "Finish" button on the previous page, thus shortening the time spent defining and starting a run.

The main advantage of the layout used on this page as compared to the CLI version of separate menus is the unified presentation to the user. With this design all pairwise alignment parameters are presented to the user on one page and in an intuitive way where options not applicable to the current state are disabled. As an added benefit, users can use these visual cues to create a better understanding of what the parameters mean and how they relate to the overall process. One specific example of this is how the set of Slow Alignment parameters used in a Pairwise alignment are nearly identical to those used in the Multiple Sequence alignment. When defining these using a CLI, there is a high risk of confusing the two sub-menus and introducing a Mode Error (discussed earlier) to that particular run. Further complicating this source of error is that the command line version of Clustal W doesn't indicate which parameters were used to perform an alignment. This is an additional feature implemented as part of the CIBA Clustal W functionality.

One known issue of the CIBA Clustal W implementation is that using the command line menu system of Clustal W allows selection of multiple output options, however the parameterized version of the Clustal W command line (what all 3rd party Clustal W GUI implementations must use) only allows one format to be selected at a time. As such, only one output format can be supported here.

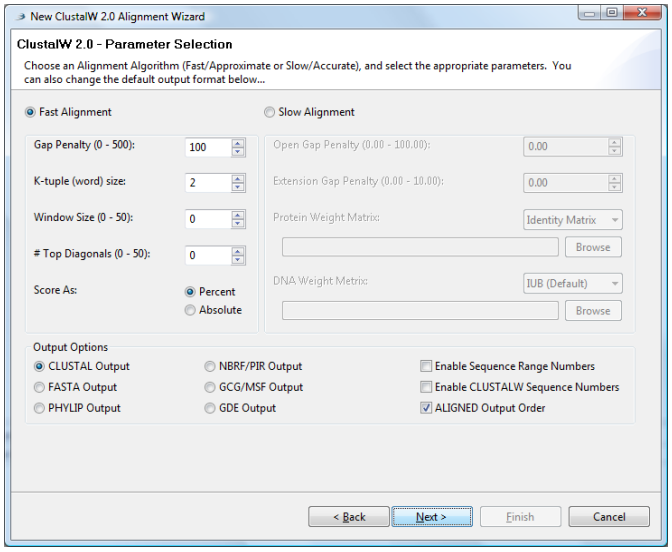


Figure 4 - CIBA: Clustal W Pairwise Alignment Parameter Selection

The last screen of this Wizard provides the Multiple Sequence Alignment parameter selection, and includes the protein/DNA weight matrices and the open/extension gap penalty definition. This last screen can also be bypassed by clicking the "Finish" button at any prior step. Future updates may include a user-definable set of defaults for each wizard page.

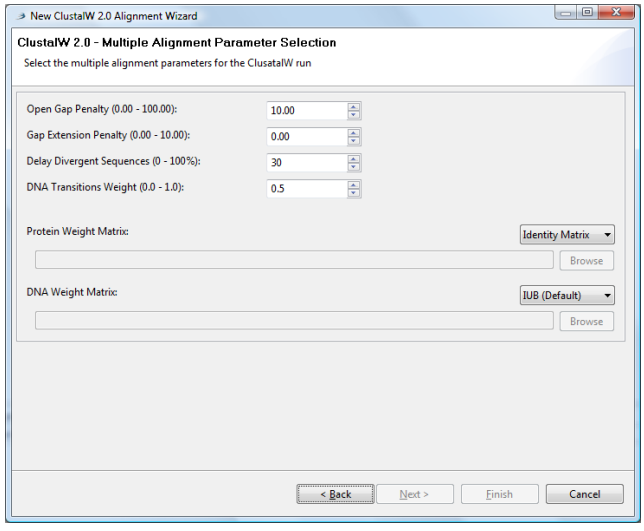


Figure 5 - CIBA: Clustal W Multiple Sequence Alignment Selection

CIBA: PHYLIP, DNADist Wizard

The design for implementing PHYLIP's DNADist program into a CIBA Wizard involved one DNADistWizard controller and one DNADistWizardPage. Similar to the Clustal W implementation, the first screen presented to the user is a Folder Selection page where the output location is selected. At the next page, the user is allowed to browse for a ".phy" alignment file, which is the resulting file from Clustal W when the PHYLIP output option is selected. The file filter can be changed to show all files, and any file that the DNADist program can use as input will work.

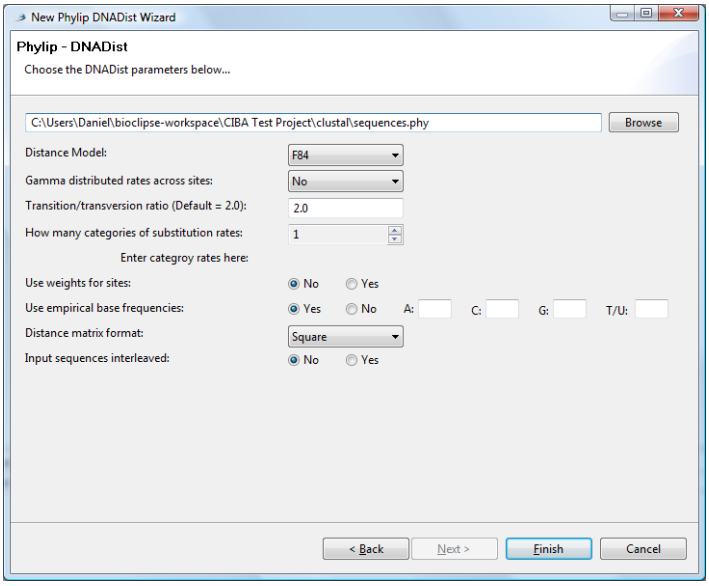


Figure 6 - CIBA: PHYLIP, DNADist Input

CIBA: PHYLIP, Neighbor Wizard

The PHYLIP, Neighbor CIBA implementation is also composed of one controller, the NeighborWizard, which contains one NeighborWizardPage. Again, the first page is for defining the output location. At the next page, the user can browse for a ".dist" distance file, which is the output from DNADist. Like the DNADist Wizard, the file filter can be changed to show all files, and any file that the Neighbor program accepts will work.

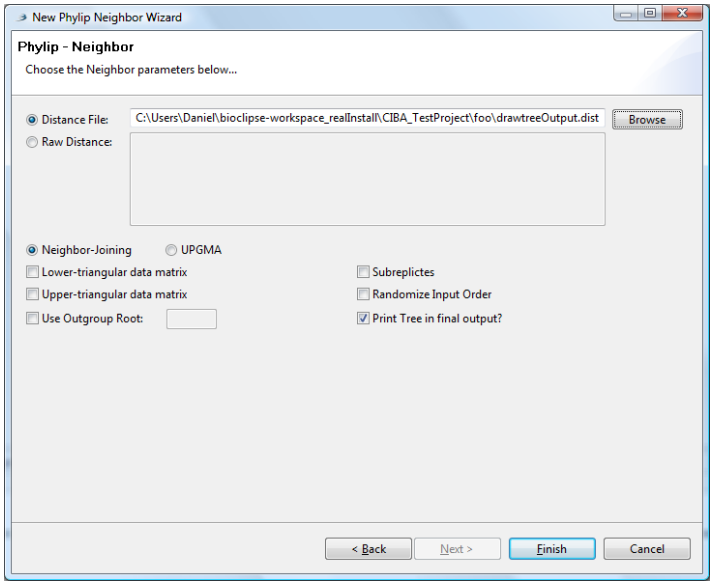


Figure 7 - CIBA: PHYLIP, Neighbor Input

CIBA: PHYLIP, Drawtree Wizard

Like the design used for DNADist, and Neighbor, the PHYLIP-Drawtree wrapper is also one Wizard controller composed of one WizardPage whose first screen is a the same Folder Selection page contained in the other wizards. At the next page, the user will be able to browse for a ".tree" tree input file, the resulting output from Neighbor. The file filter can be changed to show all files, and any file that the Neighbor program can parse is acceptable.

CIBA: Current limitations, flaws, and issues

Much effort was given to creating a perfect out-of-the-box GUI wrapper for existing CLI Bioinformatics tools, with the specific intent of creating user-friendly GUI access to command-line based applications without having to rewrite any piece of the 3rd party applications. To accomplish this necessitated the creation of a composite application whose sole purpose was to accept user-defined parameters from the GUI and broker them into the command-line arguments necessary for the program's execution. Previous sections have identified the numerous benefits of such an application, and this section will attempt to highlight some of the drawbacks.

One of the most significant disadvantages to the approach taken for CIBA is that the capabilities of the composite application are limited to the list of arguments supported by the command line program. While some may argue that tasks able to be completed with the CLI version should not differ from tasks able to be completed in the GUI version, this is not always the case. For example, when using the menu-driven command line interface for Clustal W one can enable multiple output formats for a single execution. Yet support for multiple output formats is limited to only one selection per execution when using the parameterized command line arguments to execute Clustal W. While this

is a problem for the Clustal W implementation, the method for automating PHYLIP programs at the command line addresses this disadvantage nicely. PHYLIP takes automated input from a single file, named `infile`. It then interprets the contents as line-by-line directions on how to navigate the menus, as if the input was being entered live at the keyboard. Although this addresses the issue of disparate input options, it introduces complexity to the way in which the GUI wrappers are approached.

CIBA: Evaluation

The rating system used to evaluate the subjective, objective and mixed categories of the five existing 3rd party solutions was also employed here to evaluate the new functionality created by this thesis. Before continuing, it is worth acknowledging the presence of author bias in these results. Although efforts were taken to minimize this biases (i.e. codifying distinct differences between the 1-5 ratings, pointing out flaws and limitations with the current CIBA implementations, etc), there is no perfect system in which complete elimination of bias is possible.

CIBA: Learnability

CIBA was designed to wrap command-line programs with an easier, more intuitive interface, hiding the complexity of input behind a wizard that walks the user through entry page by page. This design will improve the base learning curve for any wrapped application by bringing its disparate configuration menus into one cohesive popup dialog. Based on the definition for the category, CIBA received a 5/5 since its design facilitates use.

CIBA: Simplicity

For the same reason that CIBA scored a 5/5 in Learnability, it also received a 5/5 in the category of Simplicity.

CIBA: Informativeness

The initial parameter selection and input phase of a CIBA run promptly detects input errors and notifies the user in a prominent area underneath the wizard's title. Then, during execution of the run, CIBA opens a command line display within the Bioclipse application and streams all output from the running CLI program. It is not intelligently informative since it opens a new command line display for each run of an accessory program but lacks the ability to interpret the output the CLI program sends to this display so that it can appropriately indicate an error state. For these reasons, it was given a 4/5.

CIBA: Consistency

Since CIBA implements each command line program in a universal Wizard Framework, the look and feel between the various programs is identical. Therefore, it scored a 5/5.

CIBA: Help Documentation

Help Documentation is a difficult category to rate. In this case, the ability to provide direction and suggestions can only be as beneficial as the documentation contained in the wrapped program's own help section, plus any screen captures necessary to describe the UI. The one main advantage that a GUI implementation has over a command line help file is the ability to break apart things into smaller, more relevant units. For example, the Clustal W implementation provides help for each parameter option in the Wizard via mouse-over tool tips. It did not, however, implement page-by-

page help. As such, it was given a 3/5 for most items having help while lacking full textual descriptions and screen shots of an entire wizard page.

CIBA: Error Handling

This is another category whose results are dependent upon the accessory application being wrapped. The current implementation scored a 4/5 since the Clustal W and three PHYLIP programs have handled error situations gracefully and automatically, notifying the user as appropriate. Proper error handling, however, is limited to just the run configuration step, since recognition of error conditions produced during the execution has not been implemented.

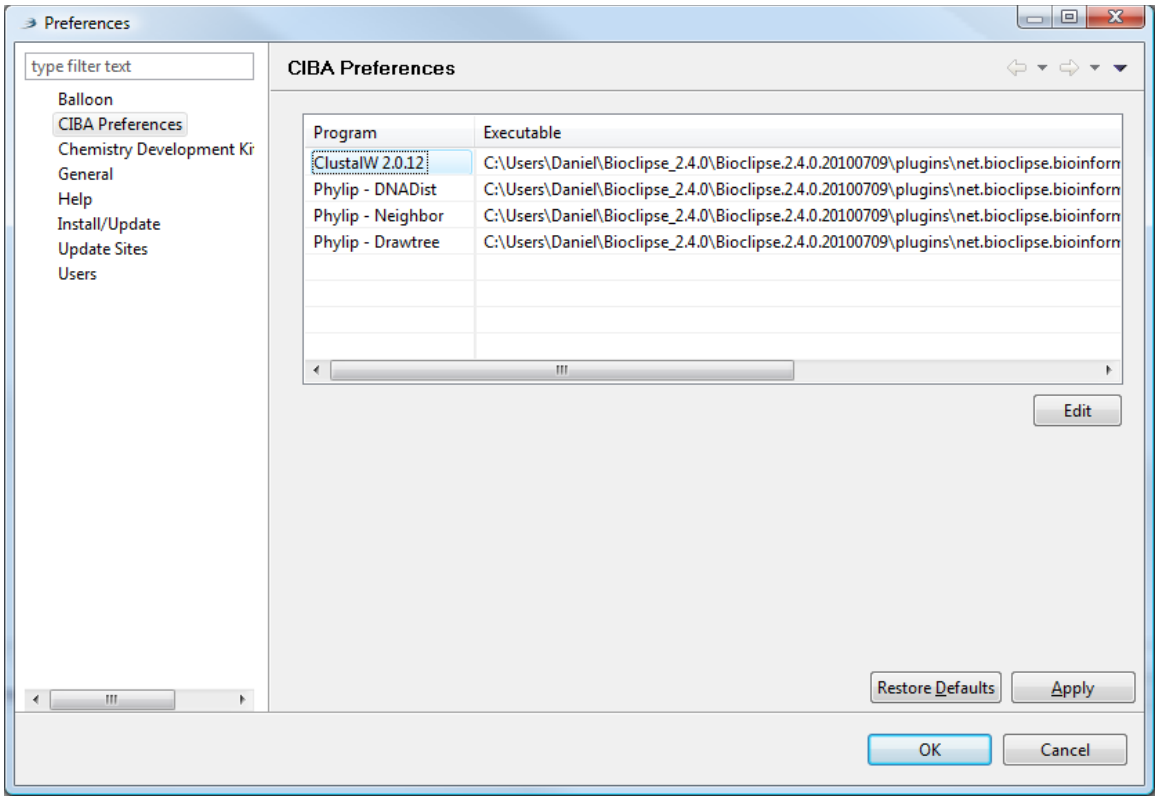
CIBA: Expandability

This is arguably the strongest category that CIBA excels in. Since it is based off an OSGi RCP application, CIBA was designed for easy, clean expansion without the risk of adversely affecting functionality in other accessory programs. It scored a 5/5.

CIBA: Comprehensiveness

CIBA currently implements the full functionality of Clustal W, with some additional features not present in the CLI version. It also implements part of the PHYLIP suite. Part of the core CIBA implementation includes a preference page (Figure 8), allowing users the flexibility to change the executable component the wizard will use. This makes upgrading the implemented wizards as simple as installing the wrapped program and changing the pointer in the preferences page. Although the additional functionality discussed above is by definition grounds for a 5/5 rating, the implementation of PHYLIP is only partially complete, so it was given a 4/5.

Figure 8 - CIBA Preferences Configuration



CIBA: Cost

CIBA is a free, open-source contribution to the Bioclipse platform. As such, it scored a 5/5.

Current State

The implementation of CIBA consists of 1 plugin containing the core functionality (the ICIBAWizard* interfaces, the CibaJob, etc), as well as the implemented Clustal W and PHYLIP wizards. The core API for CIBA, as well as the Clustal W Wizard is considered fully functional and complete. The original PHYLIP suite, on the other hand, consists of 35 separate programs, and only DNADist, Neighbor and Drawtree were implemented as part of this thesis. Completion of the remaining PHYLIP suite is considered future work, and plans are already under way.

Memory and processor usage was a large consideration at the outset of this project, however given that this plugin simply executes 3rd party software, there is minimal impact to memory and processor utilization. That being said, there is no way in Bioclipse to kill a 3rd party application started by CIBA and running in the background. Once started, users must use the OS-specific method of finding and killing the wrapped application, or wait for it to complete or time-out.

Regarding application bugs, the existing implementations have been thoroughly tested and all major bugs addressed. There are bound to be unidentified bugs in the code, and places where certain functionality does not work entirely as expected. These cases should be minimal at the current point in time.

Conclusion and Future Considerations

This paper reviewed two stand-alone Bioinformatics programs, two complex solutions that bring multiple Bioinformatics programs together into one solution, and one platform currently used in other Informatics fields that can be easily extended to include 3rd party Bioinformatics functionality. Each program was reviewed using a custom set of evaluation criteria, and the conclusion drawn that none of them provided a free, up to date, easy to use application suited for an instructional setting. To solve this problem, a highly extensible, unified and easy to use plugin was developed. Called the *Composite Interface for Bioinformatics Applications* (CIBA) plugin, it provides a simple to follow API that builds upon the core Bioclipse platform and allows developers to wrap antiquated command line programs into up to date, full-feature, cohesive GUI Wizards ideal for an instructional setting. As part of the solution, four Bioinformatics programs were implemented using the API, and are now freely available as an ad-on to Bioclipse.

In the spirit of providing the most robust and cohesive solution possible, and to encourage additional feature enhancement, the CIBA plugin is hereby released under the GNU Lesser General Public License version 3 (LGPL v3) with the explicit exception that it is not to be used in a for-profit environment[2]. Use of the GNU General Public License version 3 (GPL v3) was also considered, however due to tight integration CIBA has with 3rd party applications the less restrictive licensing option was chosen.

As for improvements, there are a few key areas where current functionality of CIBA should be improved. When the CIBA plugin is first installed, all executable locations must be defined prior to using the associated wizard for the first time. This is because the CIBA plugin relies on an existing 3rd party installation, and without user intervention the plugin is unable to locate that executable. Ideally, the CIBA implementation would include each 3rd party executable that it wraps, or at a minimum search the local disk to find it. To do that involves additional considerations ranging from longer download and install/startup times to Copyright considerations[2]. For now, the CIBA plugin will not bundle 3rd party applications, however executing a quick search on the local disk could possibly be a future enhancement.

Another area for improvement is with the implementation architecture. At the time of this writing, the CIBA implementation was done in a single plugin. Since then, the author has gained additional experience creating a custom Eclipse-based RCP application, and believes a better approach could have been taken. In this approach,

[2] The Bioclipse website indicates that Commercial Support is "coming soon", but the PHYLIP suite only grants permission to "distribute or provide access to these programs provided that this copyright notice is not removed, [and] the programs are not integrated with or called by any product or service that generates revenue"

CIBA could have been broken up into multiple plugins, each with its own preference page. The first would be the core CIBA API, interfaces, and job architecture. This would then be used by subsequent plugins that implement the CIBA architecture. In the specific implementation of this thesis, for example, there would be one CIBA Core plugin, one Clustal W plugin, and one plugin for each of the 35 PHYLIP programs. This was the initial approach taken for CIBA, however the author was unable to obtain the complete list of implementing plugins from Eclipse's OSGi framework (Equinox) without having to first open each wizard. The root cause of this was traced to the lazy loading method Eclipse uses so that plugins are only loaded on demand. The only method found to overcome the issue was to combine everything into one plugin. Going forward, CIBA implementations will be divided into separate plugins with each plugin contributing its own preference page to a core CIBA preference group.

A third area for improvement is in the help documentation. The Eclipse RCP functionality provides an easy extension point that can put a help icon and link inside each page of a Wizard. Ideally, this would include the pertinent section from the existing help file of each 3rd party application, or just open their html page directly, both of which are possible. Currently, the CIBA implementations provide hard-coded mouse-over text for each option, and the original help files for each 3rd party application can be referenced if further explanation is necessary. Ideally, all help content would be accessible from within the wizard itself, similar to how Geneious functions.

There are other areas of minor improvement, ideas for additional 3rd party integration (BLAST, Muscle, etc.), and even the possibility of creating a Facebook page to facilitate feature requests and make bug reporting easier. For the most up to date status

and a complete list of future enhancements, please see the CIBA page on the Bioclipse Developer Wiki (BioclipseWiki n.d.).

Works Cited

Attwood, Teresa K, and David J Parry-Smith. *Introduction to Bioinformatics.* Prentice Hall, 1999.

Bioclipse Features. www.bioclipse.net/Bioclipse-features (accessed April 2010).

BioclipseWiki. http://wiki.bioclipse.net/index.php?title=Main_Page (accessed April 2010).

Biomatters Ltd. www.geneious.com (accessed November 11, 2010).

—. *Geneious Plugin Writer's Guide.* http://www.geneious.com/default,1265,plugin_writers_guide.sm (accessed November 13, 2010).

Bransford, John D., Ann L. Brown, and Rodney R. Cocking. *How People Learn: Brain, Mind, Experience, and School: Expanded Edition.* Washington, D.C.: National Academy Press, 2000.

Digitales, Genes. *GUI BLAST Home Page.* 10 22, 2010. http://www.genesdigitales.com/guiblast/ (accessed 10 22, 2010).

Hagen, Joel B. "Origins of Bioinformatics." *Nature* (Macmillan Magazines, Ltd) vol. 1 (December 2000): 231-236.

Hall, Tom. *BioEdit Sequence Alignment Editor for Windows 95/98/NT/XP.* 2005. http://www.mbio.ncsu.edu/BioEdit/BioEdit.html (accessed March 2, 2009).

Kumar, Suresh. *Bioinformatics Web.* 2005. http://bioinformaticsweb.net/his.html (accessed October 22, 2010).

Larkin, M A, et al. "Clustal W and Clustal X version 2.0." *Bioinformatics* 23, no. 21 (2007): 2947-2948.

Microsoft. *Error Messages.* http://msdn.microsoft.com/en-us/library/aa511267.aspx (accessed April 22, 2012).

—. *Microsoft Launches Windows Vista and the 2007 Office System to Consumers.* http://www.microsoft.com/nz/presscentre/articles/2007/jan07_windowsvistalaunch.mspx (accessed March 2, 2009).

Needleman, Saul B., and Christian D. Wunsch. "A General Method Applicable to the Search for Similarities in the Amino Acid Sequence of Two Proteins." *Journal of Molecular Biology* 48, no. 3 (1970): 443-453.

Norman, Donald A. "Design Rules Based on Analyses of Human Error." *Communications of the ACM* vol. 26, no. 4 (1983): 254-258.

Paulin, Patrick. "Eclipse RCP Training." April 2010.

Selzer, Paul M., Richard J. Marhofer, and Andreas Rohwer. *Applied Bioinformatics, An Introduction.* Heidelberg: Springer, 2008.

Sievers, Fabian. *Clustal: Multiple Sequence Alignment.* http://www.clustal.org/omega (accessed November 12, 2011).

Sievers, Fabian, et al. "Fast, scalable generation of high-quality protein multiple sequence alignments using Clustal Omega." *Molecular Systems Biology*, October 2011: 7:539.

Spjuth, Ola. *e-mail message to author.* August 30, 2010.

Spjuth, Ola, et al. "Bioclipse 2: A scrptable integration platform for the life sciences." *BMC Bioinformatics*, 2009: 397-402.

Spjuth, Ola, et al. "Bioclipse: an open source workbench for chemo- and bioinformatics." *BMC Bioinformatics*, 2007: 59-69.

Thompson, J D, T J Gibson, F Plewniak, F Jeanmougin, and D G Higgins. "The CLUSTAL_X windows interface: flexible strategies for multiple sequence alignment aided by quality analysis tools." *Nucleic Acids Research* 25, no. 24 (December 1997): 4876-4882.